

Complexity of Equivalence and Learning for Multiplicity Tree Automata

Ines Marusic

James Worrell

Department of Computer Science

University of Oxford

Parks Road, Oxford OX1 3QD, UK

INES.MARUSIC@CS.OX.AC.UK

JBW@CS.OX.AC.UK

Abstract

We consider the complexity of equivalence and learning for multiplicity tree automata, i.e., weighted tree automata over a field. We first show that the equivalence problem is logspace equivalent to polynomial identity testing, the complexity of which is a longstanding open problem. Secondly, we derive lower bounds on the number of queries needed to learn multiplicity tree automata in Angluin’s exact learning model, over both arbitrary and fixed fields.

Habrard and Oncina (2006) give an exact learning algorithm for multiplicity tree automata, in which the number of queries is proportional to the size of the target automaton and the size of a largest counterexample, represented as a tree, that is returned by the Teacher. However, the smallest tree-counterexample may be exponential in the size of the target automaton. Thus the above algorithm does not run in time polynomial in the size of the target automaton, and has query complexity exponential in the lower bound.

Assuming a Teacher that returns minimal DAG representations of counterexamples, we give a new exact learning algorithm whose query complexity is quadratic in the target automaton size, almost matching the lower bound, and improving the best previously-known algorithm by an exponential factor.

Keywords: exact learning, query complexity, multiplicity tree automata, Hankel matrices, DAG representations of trees

1. Introduction

Trees are a natural model of structured data, including syntactic structures in natural language processing, web information extraction, and XML data on the web. Many of those applications require representing functions from trees into the real numbers. A broad class of such functions can be defined by *multiplicity tree automata*, which generalise *probabilistic tree automata*.

Multiplicity tree automata were introduced by Berstel and Reutenauer (1982) under the terminology of linear representations of tree series. They generalise classical finite tree automata by having transitions labelled by values in a field. They also generalise multiplicity word automata, introduced by Schützenberger (1961), since words are a special case of trees. Multiplicity tree automata define many natural structural properties of trees and can be used to model probabilistic processes running on trees.

Multiplicity word and tree automata have been applied to a wide variety of learning problems, including speech recognition, image processing, character recognition, and grammatical inference; see the paper of Balle and Mohri (2012) for references. A variety of

methods have been employed for learning such automata, including matrix completion and spectral methods (Balle and Mohri, 2012; Denis et al., 2014; Gybels et al., 2014) and principal components analysis (Bailly et al., 2009).

A fundamental problem concerning multiplicity tree automata is *equivalence*: given two automata, do they define the same function on trees. Seidl (1990) proved that equivalence of multiplicity tree automata is decidable in polynomial time assuming unit-cost arithmetic, and in randomised polynomial time in the usual bit-cost model. No finer analysis of the complexity of this problem exists to date. In contrast, the complexity of equivalence for classical nondeterministic word and tree automata has been completely characterised: PSPACE-complete over words (Aho et al., 1974) and EXPTIME-complete over trees (Seidl, 1990).

Our first contribution, in Section 3, is to show that the equivalence problem for multiplicity tree automata is logspace equivalent to polynomial identity testing, i.e., the problem of deciding whether a polynomial given as an arithmetic circuit is zero. The latter is known to be solvable in randomised polynomial time (DeMillo and Lipton, 1978; Schwartz, 1980; Zippel, 1979), whereas solving it in deterministic polynomial time is a well-studied and longstanding open problem (see Arora and Barak, 2009).

Equivalence is closely connected to the problem of learning in the *exact learning model* of Angluin (1988). In this model, a Learner actively collects information about the target function from a Teacher through *membership queries*, which ask for the value of the function on a specific input, and *equivalence queries*, which suggest a hypothesis to which the Teacher provides a counterexample if one exists. A class of functions \mathcal{C} is *exactly learnable* if there exists an exact learning algorithm such that for any function $f \in \mathcal{C}$, the Learner identifies f using polynomially many membership and equivalence queries in the size of a shortest representation of f and the size of a largest counterexample returned by the Teacher during the execution of the algorithm. The exact learning model is an important theoretical model of the learning process. It is well-known that learnability in the exact learning model also implies learnability in the PAC model with membership queries (Valiant, 1985).

One of the earliest results about the exact learning model was the proof of Angluin (1987) that deterministic finite automata are learnable. This result was generalised by Drewes and Högberg (2003) to show exact learnability of deterministic finite tree automata, generalising also a result of Sakakibara (1990) on the exact learnability of context-free grammars from structural data.

Exact learnability of multiplicity automata has also been extensively studied. Beimel et al. (2000) show that multiplicity word automata can be learned efficiently, and apply this to learn various classes of DNF formulae and polynomials. These results were generalised by Klivans and Shpilka (2006) to show exact learnability of restricted algebraic branching programs and noncommutative set-multilinear arithmetic formulae. Bisht et al. (2006) give an almost tight (up to a \log factor) lower bound for the number of queries made by any exact learning algorithm for the class of multiplicity word automata. Finally, Habrard and Oncina (2006) give an algorithm for learning multiplicity tree automata in the exact model.

Our first contribution on learning multiplicity tree automata, in Section 5, is to give lower bounds on the number of queries needed to learn multiplicity tree automata in the exact learning model, both for the case of an arbitrary and a fixed underlying field. The bound in the latter case is proportional to the automaton size for trees of a fixed maximal

branching degree. To the best of our knowledge, these are the first lower bounds on the query complexity of exactly learning of multiplicity tree automata.

Consider a target multiplicity tree automaton whose minimal representation A has n states. The algorithm of Habrard and Oncina (2006) makes at most n equivalence queries and number of membership queries proportional to $|A| \cdot s$, where $|A|$ is the size of A and s is the size of a largest counterexample returned by the Teacher. Since this algorithm assumes that the Teacher returns counterexamples represented explicitly as trees, s can be exponential in $|A|$, even for a Teacher that returns counterexamples of minimal size (see Example 1). This observation reveals an exponential gap between the query complexity of the algorithm of Habrard and Oncina (2006) and our above-mentioned lower bound, which is only linear in $|A|$. Another consequence is that the worst-case time complexity of this algorithm is exponential in the size of the target automaton.

Given two inequivalent multiplicity tree automata with n states in total, the algorithm of Seidl (1990) produces a subtree-closed set of trees of cardinality at most n that contains a tree on which the automata differ. It follows that the counterexample contained in this set has at most n subtrees, and hence can be represented as a DAG with at most n vertices. Thus in the context of exact learning it is natural to consider a Teacher that can return succinctly represented counterexamples, i.e., trees represented as DAGs.

Tree automata that run on DAG representations of finite trees were first introduced by Charatonik (1999) as extensions of ordinary tree automata, and were further studied by Anantharaman et al. (2005). The automata considered by Charatonik (1999) and Anantharaman et al. (2005) run on fully-compressed DAGs. Fila and Anantharaman (2006) extended this definition by introducing tree automata that run on DAGs that may be partially compressed. In this paper, we employ the latter framework in the context of learning multiplicity automata.

In Section 4, we present a new exact learning algorithm for multiplicity tree automata that achieves the same bound in the number of equivalence queries as the algorithm of Habrard and Oncina (2006), while using number of membership queries quadratic in the target automaton size and linear in the counterexample size, even when counterexamples are given succinctly. Assuming that the Teacher provides minimal DAG representations of counterexamples, our algorithm therefore makes quadratically many queries in the target size. This is exponentially fewer queries than the best previously-known algorithm (Habrard and Oncina, 2006) and within a linear factor of the above-mentioned lower bound. Furthermore, our algorithm performs a quadratic number of arithmetic operations in the size of the target automaton, and can be implemented in randomised polynomial time in the Turing model.

Like the algorithm of Habrard and Oncina (2006), our algorithm constructs a Hankel matrix of the target automaton. However on receiving a counterexample tree z , the former algorithm adds a new column to the Hankel matrix for every suffix of z , while our algorithm adds (at most) one new row for each subtree of z . Crucially the number of suffixes may be exponential in the size of a DAG representation of z , whereas the number of subtrees is only linear in the size of a DAG representation.

An extended abstract (Marusic and Worrell, 2014) of this work appeared in the proceedings of MFCS 2014. The current paper contains full proofs of all results reported there, the

formal definition of multiplicity tree automata running on DAGs, and a refined complexity analysis of the learning algorithm.

2. Preliminaries

Let \mathbb{N} and \mathbb{N}_0 denote the set of all positive and non-negative integers, respectively. Let $n \in \mathbb{N}$. We write $[n]$ for the set $\{1, 2, \dots, n\}$ and I_n for the identity matrix of order n . For every $i \in [n]$, we write e_i for the i^{th} n -dimensional coordinate row vector.

For any matrix A , we write A_i for its i^{th} row, A^j for its j^{th} column, and $A_{i,j}$ for its $(i, j)^{\text{th}}$ entry. Given non-empty subsets I and J of the rows and columns of A , respectively, we write $A_{I,J}$ for the submatrix $(A_{i,j})_{i \in I, j \in J}$ of A . For singletons, we write simply $A_{i,J} := A_{\{i\},J}$ and $A_{I,j} := A_{I,\{j\}}$. Let $n_1, \dots, n_k \in \mathbb{N}$, and let A be a matrix with $n_1 \cdot \dots \cdot n_k$ rows. For every $(i_1, \dots, i_k) \in [n_1] \times \dots \times [n_k]$, we write $A_{(i_1, \dots, i_k)}$ for the $(\sum_{l=1}^{k-1} (i_l - 1) \cdot (\prod_{p=l+1}^k n_p) + i_k)^{\text{th}}$ row of A .

Given a set V , we denote by V^* the set of all finite ordered tuples of elements from V . For any subset $S \subseteq V$, the *characteristic function* of S (relative to V) is the function $\chi_S : V \rightarrow \{0, 1\}$ such that $\chi_S(x) = 1$ if $x \in S$, and $\chi_S(x) = 0$ otherwise.

2.1 Kronecker Product

Let A be a matrix of dimension $m_1 \times n_1$ and B a matrix of dimension $m_2 \times n_2$. The *Kronecker product* of A by B , written as $A \otimes B$, is a matrix of dimension $m_1 m_2 \times n_1 n_2$ where $(A \otimes B)_{(i_1, i_2), (j_1, j_2)} = A_{i_1, j_1} \cdot B_{i_2, j_2}$ for every $i_1 \in [m_1]$, $i_2 \in [m_2]$, $j_1 \in [n_1]$, $j_2 \in [n_2]$.

The Kronecker product is bilinear, associative, and has the following *mixed-product property*: For any matrices A, B, C, D such that products $A \cdot C$ and $B \cdot D$ are defined, it holds that $(A \otimes B) \cdot (C \otimes D) = (A \cdot C) \otimes (B \cdot D)$.

Let $k \in \mathbb{N}$ and A_1, \dots, A_k be matrices such that for every $l \in [k]$, A_l has n_l rows. It can easily be shown using induction on k that for every $(i_1, \dots, i_k) \in [n_1] \times \dots \times [n_k]$, it holds that

$$(A_1 \otimes \dots \otimes A_k)_{(i_1, \dots, i_k)} = (A_1)_{i_1} \otimes \dots \otimes (A_k)_{i_k}. \quad (1)$$

We write $\bigotimes_{l=1}^k A_l := A_1 \otimes \dots \otimes A_k$.

For every $k \in \mathbb{N}_0$ we define the k -fold *Kronecker power* of a matrix A , written as $A^{\otimes k}$, inductively by $A^{\otimes 0} = I_1$ and $A^{\otimes k} = A^{\otimes (k-1)} \otimes A$ for $k \geq 1$.

Let $k \in \mathbb{N}_0$. For any matrices A, B of appropriate dimensions, we have

$$(A \otimes B)^k = A^k \otimes B^k. \quad (2)$$

For any matrices A_1, \dots, A_k and B_1, \dots, B_k where product $A_l \cdot B_l$ is defined for every $l \in [k]$, we have

$$(A_1 \otimes \dots \otimes A_k) \cdot (B_1 \otimes \dots \otimes B_k) = (A_1 \cdot B_1) \otimes \dots \otimes (A_k \cdot B_k). \quad (3)$$

Equations (2) and (3) follow easily from the mixed-product property by induction on k .

2.2 Finite Trees

A *ranked alphabet* is a tuple (Σ, rk) where Σ is a nonempty finite set of symbols and $rk : \Sigma \rightarrow \mathbb{N}_0$ is a function. Ranked alphabet (Σ, rk) is often written Σ for short. For every $k \in \mathbb{N}_0$, we define the set of all k -ary symbols $\Sigma_k := rk^{-1}(\{k\})$. If $\sigma \in \Sigma_k$ then we say that σ has *rank* (or *arity*) k . We say that Σ has *rank* m if $m = \max\{rk(\sigma) : \sigma \in \Sigma\}$.

The set of Σ -trees (*trees* for short), written as T_Σ , is the smallest set T satisfying the following two conditions: (i) $\Sigma_0 \subseteq T$; and (ii) if $k \geq 1$, $\sigma \in \Sigma_k$, $t_1, \dots, t_k \in T$ then $\sigma(t_1, \dots, t_k) \in T$. Given a Σ -tree t , a *subtree* of t is a Σ -tree consisting of a node in t and all of its descendants in t . The set of all subtrees of t is denoted by $Sub(t)$.

Let Σ be a ranked alphabet and \mathbb{F} be a field. A *tree series* over Σ with coefficients in \mathbb{F} is a function $f : T_\Sigma \rightarrow \mathbb{F}$. For every $t \in T_\Sigma$, we call $f(t)$ the *coefficient* of t in f . The set of all tree series over Σ with coefficients in \mathbb{F} is denoted by $\mathbb{F}\langle\langle T_\Sigma \rangle\rangle$.

We define the tree series *height*, *size*, $\#_\sigma \in \mathbb{Q}\langle\langle T_\Sigma \rangle\rangle$ where $\sigma \in \Sigma$, as follows: (i) if $t \in \Sigma_0$ then $height(t) = 0$, $size(t) = 1$, $\#_\sigma(t) = \chi_{\{t=\sigma\}}$; and (ii) if $t = a(t_1, \dots, t_k)$ where $k \geq 1$, $a \in \Sigma_k$, $t_1, \dots, t_k \in T_\Sigma$ then $height(t) = 1 + \max_{i \in [k]} height(t_i)$, $size(t) = 1 + \sum_{i \in [k]} size(t_i)$, $\#_\sigma(t) = \chi_{\{a=\sigma\}} + \sum_{i \in [k]} \#_\sigma(t_i)$, respectively. For every $n \in \mathbb{N}_0$, we define the sets $T_\Sigma^{<n} := \{t \in T_\Sigma : height(t) < n\}$, $T_\Sigma^n := \{t \in T_\Sigma : height(t) = n\}$, and $T_\Sigma^{\leq n} := T_\Sigma^{<n} \cup T_\Sigma^n$.

Let \square be a nullary symbol not contained in Σ . The set C_Σ of Σ -contexts (*contexts* for short) is the set of $(\{\square\} \cup \Sigma)$ -trees in which \square occurs exactly once. The *concatenation* of $c \in C_\Sigma$ and $t \in T_\Sigma \dot{\cup} C_\Sigma$, written as $c[t]$, is the tree obtained by substituting t for \square in c . A *suffix* of a Σ -tree t is a Σ -context c such that $t = c[t']$ for some Σ -tree t' . The *Hankel matrix* of a tree series $f \in \mathbb{F}\langle\langle T_\Sigma \rangle\rangle$ is the matrix $H : T_\Sigma \times C_\Sigma \rightarrow \mathbb{F}$ such that $H_{t,c} = f(c[t])$ for every $t \in T_\Sigma$ and $c \in C_\Sigma$.

2.3 Multiplicity Tree Automata

Let \mathbb{F} be a field. An \mathbb{F} -multiplicity tree automaton (\mathbb{F} -MTA) is a quadruple $A = (n, \Sigma, \mu, \gamma)$ which consists of the *dimension* $n \in \mathbb{N}_0$ representing the number of states, a ranked alphabet Σ , the *tree representation* $\mu = \{\mu(\sigma) : \sigma \in \Sigma\}$ where for every symbol $\sigma \in \Sigma$, matrix $\mu(\sigma) \in \mathbb{F}^{n^{rk(\sigma)} \times n}$ represents the *transition matrix* associated to σ , and the *final weight vector* $\gamma \in \mathbb{F}^{n \times 1}$. The *size* of the automaton A , written as $|A|$, is defined as

$$|A| := \sum_{\sigma \in \Sigma} n^{rk(\sigma)+1} + n.$$

That is, the size of A is the total number of entries in all transition matrices and the final weight vector.¹

We extend the tree representation μ from Σ to T_Σ by defining

$$\mu(\sigma(t_1, \dots, t_k)) := (\mu(t_1) \otimes \dots \otimes \mu(t_k)) \cdot \mu(\sigma)$$

for every $\sigma \in \Sigma_k$ and $t_1, \dots, t_k \in T_\Sigma$. The tree series $\|A\| \in \mathbb{F}\langle\langle T_\Sigma \rangle\rangle$ recognised by A is defined by $\|A\|(t) = \mu(t) \cdot \gamma$ for every $t \in T_\Sigma$. Note that a 0-dimensional automaton

1. We measure size assuming explicit rather than sparse representations of the transition matrices and final weight vector because minimal automata are only unique up to change of basis (see Theorem 4).

necessarily recognises a zero tree series. Two automata A_1, A_2 are said to be *equivalent* if $\|A_1\| \equiv \|A_2\|$.

We further extend μ from T_Σ to C_Σ by treating \square as a unary symbol and defining $\mu(\square) := I_n$. This allows to define $\mu(c) \in \mathbb{F}^{n \times n}$ for every $c = \sigma(t_1, \dots, t_k) \in C_\Sigma$ inductively as $\mu(c) := (\mu(t_1) \otimes \dots \otimes \mu(t_k)) \cdot \mu(\sigma)$. It is easy to see that $\mu(c[t]) = \mu(t) \cdot \mu(c)$ for every $t \in T_\Sigma$ and $c \in C_\Sigma$.

Let $A_1 = (n_1, \Sigma, \mu_1, \gamma_1)$ and $A_2 = (n_2, \Sigma, \mu_2, \gamma_2)$ be two \mathbb{F} -multiplicity tree automata. The *product* of A_1 by A_2 , written as $A_1 \times A_2$, is the \mathbb{F} -multiplicity tree automaton (n, Σ, μ, γ) where:

- $n = n_1 \cdot n_2$;
- If $\sigma \in \Sigma_k$ then $\mu(\sigma) = P_k \cdot (\mu_1(\sigma) \otimes \mu_2(\sigma))$ where P_k is a permutation matrix of order $(n_1 \cdot n_2)^k$ uniquely defined (see Remark 1 below) by

$$(u_1 \otimes \dots \otimes u_k) \otimes (v_1 \otimes \dots \otimes v_k) = ((u_1 \otimes v_1) \otimes \dots \otimes (u_k \otimes v_k)) \cdot P_k \quad (4)$$

for all $u_1, \dots, u_k \in \mathbb{F}^{1 \times n_1}$ and $v_1, \dots, v_k \in \mathbb{F}^{1 \times n_2}$;

- $\gamma = \gamma_1 \otimes \gamma_2$.

Remark 1 We argue that for every rank k of a symbol in Σ , matrix P_k is well-defined by Equation (4). In order to do this, it suffices to show that P_k is well-defined on a set of basis vectors of $\mathbb{F}^{1 \times n_1}$ and $\mathbb{F}^{1 \times n_2}$ and then extend linearly. To that end, let $(e_i^1)_{i \in [n_1]}$ and $(e_j^2)_{j \in [n_2]}$ be bases of $\mathbb{F}^{1 \times n_1}$ and $\mathbb{F}^{1 \times n_2}$, respectively. Let us define sets of vectors

$$E_1 := \{(e_{i_1}^1 \otimes \dots \otimes e_{i_k}^1) \otimes (e_{j_1}^2 \otimes \dots \otimes e_{j_k}^2) : i_1, \dots, i_k \in [n_1], j_1, \dots, j_k \in [n_2]\}$$

and

$$E_2 := \{(e_{i_1}^1 \otimes e_{j_1}^2) \otimes \dots \otimes (e_{i_k}^1 \otimes e_{j_k}^2) : i_1, \dots, i_k \in [n_1], j_1, \dots, j_k \in [n_2]\}.$$

Then, E_1 and E_2 are two bases of the vector space $\mathbb{F}^{1 \times n_1 n_2}$. Therefore, P_k is well-defined as an invertible matrix mapping basis E_1 to basis E_2 .

Essentially the same product construction as in the proof of the first part of the following proposition is given by Berstel and Reutenauer (1982, Proposition 5.1) in the language of linear representations of tree series rather than multiplicity tree automata.

Proposition 2 Let A_1 and A_2 be \mathbb{Q} -multiplicity tree automata over a ranked alphabet Σ . Then, for every $t \in T_\Sigma$ it holds that $\|A_1 \times A_2\|(t) = \|A_1\|(t) \cdot \|A_2\|(t)$. Furthermore, automaton $A_1 \times A_2$ can be computed from A_1 and A_2 in logarithmic space.

Proof Let $A_1 = (n_1, \Sigma, \mu_1, \gamma_1)$, $A_2 = (n_2, \Sigma, \mu_2, \gamma_2)$, and $A_1 \times A_2 = (n, \Sigma, \mu, \gamma)$. First we show that for any $t \in T_\Sigma$,

$$\mu(t) = \mu_1(t) \otimes \mu_2(t). \quad (5)$$

We prove that Equation (5) holds for all $t \in T_\Sigma$ using induction on $\text{height}(t)$. The base case $t = \sigma \in \Sigma_0$ holds immediately by definition since $P_0 = I_1$. For the induction step, let

$h \in \mathbb{N}_0$ and assume that Equation (5) holds for every $t \in T_{\Sigma}^{\leq h}$. Take any $t \in T_{\Sigma}^{h+1}$. Then $t = \sigma(t_1, \dots, t_k)$ for some $k \geq 1$, $\sigma \in \Sigma_k$, and $t_1, \dots, t_k \in T_{\Sigma}^{\leq h}$. By induction hypothesis, Equation (4), and the mixed-product property of Kronecker product we now have

$$\begin{aligned} \mu(t) &= (\mu(t_1) \otimes \dots \otimes \mu(t_k)) \cdot \mu(\sigma) \\ &= ((\mu_1(t_1) \otimes \mu_2(t_1)) \otimes \dots \otimes (\mu_1(t_k) \otimes \mu_2(t_k))) \cdot P_k \cdot (\mu_1(\sigma) \otimes \mu_2(\sigma)) \\ &= ((\mu_1(t_1) \otimes \dots \otimes \mu_1(t_k)) \otimes (\mu_2(t_1) \otimes \dots \otimes \mu_2(t_k))) \cdot (\mu_1(\sigma) \otimes \mu_2(\sigma)) \\ &= ((\mu_1(t_1) \otimes \dots \otimes \mu_1(t_k)) \cdot \mu_1(\sigma)) \otimes ((\mu_2(t_1) \otimes \dots \otimes \mu_2(t_k)) \cdot \mu_2(\sigma)) \\ &= \mu_1(t) \otimes \mu_2(t). \end{aligned}$$

This completes the proof of Equation (5) for all $t \in T_{\Sigma}$ by induction. For every $t \in T_{\Sigma}$, we now have

$$\begin{aligned} \|A_1 \times A_2\|(t) &= \mu(t) \cdot \gamma = (\mu_1(t) \otimes \mu_2(t)) \cdot (\gamma_1 \otimes \gamma_2) \\ &= (\mu_1(t) \cdot \gamma_1) \otimes (\mu_2(t) \cdot \gamma_2) = \|A_1\|(t) \otimes \|A_2\|(t) = \|A_1\|(t) \cdot \|A_2\|(t). \end{aligned}$$

We conclude by noting that automaton $A_1 \times A_2$ can be computed from A_1 and A_2 by an algorithm that maintains a constant number of pointers, therefore requiring only logarithmic space. \blacksquare

A tree series f is called *recognisable* if it is recognised by some multiplicity tree automaton; such an automaton is called an *MTA-representation* of f . An MTA-representation of f that has the smallest dimension is called *minimal*. The set of all recognisable tree series in $\mathbb{F}\langle\langle T_{\Sigma} \rangle\rangle$ is denoted by $\text{Rec}(\Sigma, \mathbb{F})$.

The following result was first shown by Bozapalidis and Louscou-Bozapalidou (1983); an essentially equivalent result was later shown by Habrard and Oncina (2006).

Theorem 3 (Bozapalidis and Louscou-Bozapalidou, 1983) *Let Σ be a ranked alphabet and \mathbb{F} be a field. Let $f \in \mathbb{F}\langle\langle T_{\Sigma} \rangle\rangle$ and let H be the Hankel matrix of f . It holds that $f \in \text{Rec}(\Sigma, \mathbb{F})$ if and only if H has finite rank over \mathbb{F} . In case $f \in \text{Rec}(\Sigma, \mathbb{F})$, the dimension of a minimal MTA-representation of f is $\text{rank}(H)$ over \mathbb{F} .*

The following result by Bozapalidis and Alexandrakis (1989, Proposition 4) states that for any recognisable tree series, its minimal MTA-representation is unique up to change of basis.

Theorem 4 (Bozapalidis and Alexandrakis, 1989) *Let Σ be a ranked alphabet and \mathbb{F} be a field. Let $f \in \text{Rec}(\Sigma, \mathbb{F})$ and let r be the rank (over \mathbb{F}) of the Hankel matrix of f . Let $A_1 = (r, \Sigma, \mu_1, \gamma_1)$ be an MTA-representation of f . Given an \mathbb{F} -multiplicity tree automaton $A_2 = (r, \Sigma, \mu_2, \gamma_2)$, it holds that A_2 recognises f if and only if there exists an invertible matrix $U \in \mathbb{F}^{r \times r}$ such that $\gamma_2 = U \cdot \gamma_1$ and $\mu_2(\sigma) = U^{\otimes rk(\sigma)} \cdot \mu_1(\sigma) \cdot U^{-1}$ for every $\sigma \in \Sigma$.*

2.4 DAG Representations of Finite Trees

Let Σ be a ranked alphabet. A *DAG representation* of a Σ -tree (Σ -DAG or DAG for short) is a rooted directed acyclic ordered multigraph whose nodes are labelled with symbols from

Σ such that the outdegree of each node is equal to the rank of the symbol it is labelled with. Formally a Σ -DAG consists of a set of nodes V , for each node $v \in V$ a list of successors $\text{succ}(v) \in V^*$, and a node labelling $\lambda : V \rightarrow \Sigma$ where for each node $v \in V$ it holds that $\lambda(v) \in \Sigma_{|\text{succ}(v)|}$. Note that Σ -trees are a subclass of Σ -DAGs.

Let G be a Σ -DAG. The *size* of G , denoted by $\text{size}(G)$, is the number of nodes in G . The *height* of G , denoted by $\text{height}(G)$, is the length of a longest directed path in G . For any node v in G , the *sub-DAG* of G *rooted at* v , denoted by $G|_v$, is the Σ -DAG consisting of the node v and all of its descendants in G . Clearly, if a node v_0 is the root of G then $G|_{v_0} = G$. The set $\{G|_v : v \text{ is a node in } G\}$ of all the sub-DAGs of G is denoted by $\text{Sub}(G)$.

For any Σ -DAG G , we define its *unfolding* into a Σ -tree, denoted by $\text{unfold}(G)$, inductively as follows: If the root of G is labelled with a symbol σ and has the list of successors v_1, \dots, v_k , then

$$\text{unfold}(G) = \sigma(\text{unfold}(G|_{v_1}), \dots, \text{unfold}(G|_{v_k})).$$

It is easy to see that the following proposition holds.

Proposition 5 *If G is a Σ -DAG, then $\text{Sub}(\text{unfold}(G)) = \text{unfold}[\text{Sub}(G)]$.*

Because a context has exactly one occurrence of the symbol \square , any *DAG representation* of a Σ -context is a $(\{\square\} \cup \Sigma)$ -DAG that has a unique path from the root to the (unique) \square -labelled node. The *concatenation* of a DAG K , representing a Σ -context, and a Σ -DAG G , denoted by $K[G]$, is the Σ -DAG obtained by substituting the root of G for \square in K .

Proposition 6 *Let K be a DAG representation of a Σ -context, and let G be a Σ -DAG. Then, $\text{unfold}(K[G]) = \text{unfold}(K)[\text{unfold}(G)]$.*

Proof The proof is by induction on $\text{height}(K)$. For the base case $\text{height}(K) = 0$, we have that $K = \square$ and therefore $\text{unfold}(\square[G]) = \text{unfold}(G) = \text{unfold}(\square)[\text{unfold}(G)]$ for any Σ -DAG G .

For the induction step, let $h \in \mathbb{N}_0$ and assume that the proposition holds if $\text{height}(K) \leq h$. Let K be a DAG representation of a Σ -context such that $\text{height}(K) = h + 1$. Let the root of K have label σ and list of successors v_1, \dots, v_k . By definition, there is a unique path in K going from the root to the \square -labelled node. Without loss of generality, we can assume that the \square -labelled node is a successor of v_1 . Take an arbitrary Σ -DAG G . Since $\text{height}(K|_{v_1}) \leq h$, we have by the induction hypothesis that

$$\begin{aligned} \text{unfold}(K[G]) &= \sigma(\text{unfold}(K|_{v_1}[G]), \text{unfold}(K|_{v_2}), \dots, \text{unfold}(K|_{v_k})) \\ &= \sigma(\text{unfold}(K|_{v_1})[\text{unfold}(G)], \text{unfold}(K|_{v_2}), \dots, \text{unfold}(K|_{v_k})) \\ &= \sigma(\text{unfold}(K|_{v_1}), \text{unfold}(K|_{v_2}), \dots, \text{unfold}(K|_{v_k}))[\text{unfold}(G)] \\ &= \text{unfold}(K)[\text{unfold}(G)]. \end{aligned}$$

This completes the proof. ■

2.5 Multiplicity Tree Automata on DAGs

In this section, we introduce the notion of a multiplicity tree automaton on DAGs. To the best of our knowledge, this notion has not been studied before.

Let \mathbb{F} be a field, and $A = (n, \Sigma, \mu, \gamma)$ be an \mathbb{F} -multiplicity tree automaton. The computation of the automaton A on a Σ -DAG $G = (V, E)$ is defined as follows: A *run* of A on G is a mapping $\rho : \text{Sub}(G) \rightarrow \mathbb{F}^n$ such that for every node $v \in V$, if v is labelled with σ and has the list of successors $\text{succ}(v) = v_1, \dots, v_k$ then

$$\rho(G|_v) = (\rho(G|_{v_1}) \otimes \dots \otimes \rho(G|_{v_k})) \cdot \mu(\sigma).$$

Automaton A assigns to G a *weight* $\|A\|(G) \in \mathbb{F}$ where $\|A\|(G) = \rho(G) \cdot \gamma$.

In the following proposition, we show that the weight assigned by a multiplicity tree automaton to a DAG is equal to the weight assigned to its tree unfolding.

Proposition 7 *Let \mathbb{F} be a field, and $A = (n, \Sigma, \mu, \gamma)$ be an \mathbb{F} -multiplicity tree automaton. For any Σ -DAG G , it holds that $\rho(G) = \mu(\text{unfold}(G))$ and $\|A\|(G) = \|A\|(\text{unfold}(G))$.*

Proof Let V be the set of nodes of G . First we show that for every $v \in V$,

$$\rho(G|_v) = \mu(\text{unfold}(G|_v)). \quad (6)$$

The proof is by induction on $\text{height}(G|_v)$. For the base case, let $\text{height}(G|_v) = 0$. This implies that $G|_v = \sigma \in \Sigma_0$. Therefore, by definition we have that

$$\rho(G|_v) = \mu(\sigma) = \mu(\text{unfold}(\sigma)) = \mu(\text{unfold}(G|_v)).$$

For the induction step, let $h \in \mathbb{N}_0$ and assume that Equation (6) holds for every $v \in V$ such that $\text{height}(G|_v) \leq h$. Take any $v \in V$ such that $\text{height}(G|_v) = h + 1$. Let the root of $G|_v$ be labelled with a symbol σ and have list of successors $\text{succ}(v) = v_1, \dots, v_k$. Then for every $j \in [k]$, we have that $\text{height}(G|_{v_j}) \leq h$ and thus $\rho(G|_{v_j}) = \mu(\text{unfold}(G|_{v_j}))$ holds by the induction hypothesis. This implies that

$$\begin{aligned} \rho(G|_v) &= (\rho(G|_{v_1}) \otimes \dots \otimes \rho(G|_{v_k})) \cdot \mu(\sigma) \\ &= (\mu(\text{unfold}(G|_{v_1})) \otimes \dots \otimes \mu(\text{unfold}(G|_{v_k}))) \cdot \mu(\sigma) \\ &= \mu(\sigma(\text{unfold}(G|_{v_1}), \dots, \text{unfold}(G|_{v_k}))) \\ &= \mu(\text{unfold}(G|_v)) \end{aligned}$$

which completes the proof of Equation (6) for all $v \in V$ by induction.

Taking v to be the root of G , we get from Equation (6) that $\rho(G) = \mu(\text{unfold}(G))$. Therefore, $\|A\|(G) = \rho(G) \cdot \gamma = \mu(\text{unfold}(G)) \cdot \gamma = \|A\|(\text{unfold}(G))$. \blacksquare

Example 1 *Let $\Sigma = \{\sigma_0, \sigma_2\}$ be a ranked alphabet such that $\text{rk}(\sigma_0) = 0$ and $\text{rk}(\sigma_2) = 2$. Take any $n \in \mathbb{N}$. Let t_n , depicted in Figure 1, be the perfect binary Σ -tree of height $n - 1$. Note that $\text{size}(t_n) = O(2^n)$. Define a \mathbb{Q} -MTA $A = (n, \Sigma, \mu, e_1)$ such that $\mu(\sigma_0) = e_n \in \mathbb{F}^{1 \times n}$ and $\mu(\sigma_2) \in \mathbb{F}^{n^2 \times n}$ where $\mu(\sigma_2)_{(i+1, i+1), i} = 1$ for every $i \in [n - 1]$, and all other entries of $\mu(\sigma_2)$ are zero. It is easy to see that $\|A\|(t_n) = 1$ and $\|A\|(t) = 0$ for every $t \in T_\Sigma \setminus \{t_n\}$.*

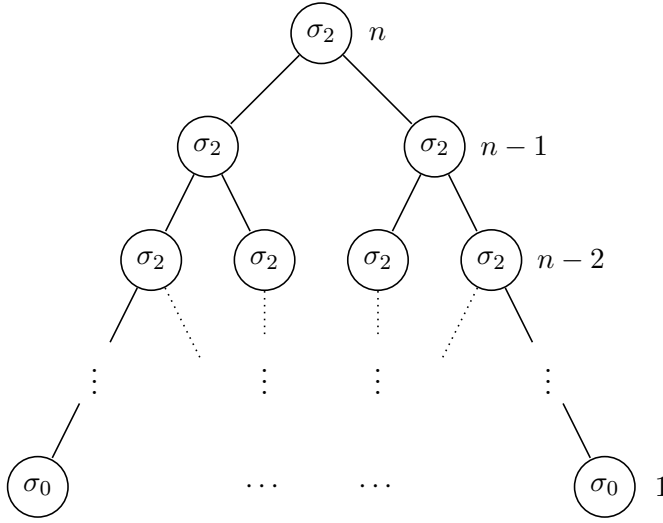


Figure 1: Tree t_n

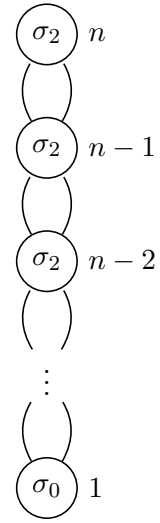


Figure 2: DAG G_n

Let B be the 0-dimensional \mathbb{Q} -MTA over Σ (so that $\|B\| \equiv 0$). Suppose we were to check whether automata A and B are equivalent. Then the only counterexample to their equivalence, namely the tree t_n , has size $O(2^n)$. Note, however, that t_n has an exponentially more succinct DAG representation G_n , given in Figure 2.

2.6 Arithmetic Circuits

An *arithmetic circuit* is a finite directed acyclic vertex-labelled multigraph whose vertices, called *gates*, have indegree 0 or 2. Vertices of indegree 0 are called *input gates* and are labelled with a constant 0 or 1, or a variable from the set $\{x_i : i \in \mathbb{N}\}$. Vertices of indegree 2 are called *internal gates* and are labelled with an arithmetic operation $+$, \times , or $-$. We assume that there is a unique gate with outdegree 0 called the *output gate*. An arithmetic circuit is called *variable-free* if all input gates are labelled with 0 or 1.

Given two gates u and v of an arithmetic circuit C , we call u a *child* of v if (u, v) is a directed edge in C . The *size* of C is the number of gates in C . The *height* of a gate v in C , written as $\text{height}(v)$, is the length of a longest directed path from an input gate to v . The *height* of C is the maximal height of a gate in C .

An arithmetic circuit C computes a polynomial over the integers as follows: An input gate of C labelled with $\alpha \in \{0, 1\} \cup \{x_i : i \in \mathbb{N}\}$ computes the polynomial α . An internal gate of C labelled with $*$ $\in \{+, \times, -\}$ computes the polynomial $p_1 * p_2$ where p_1 and p_2 are the polynomials computed by its children. For any gate v in C , we write f_v for the polynomial computed by v . The *output* of C , written f_C , is the polynomial computed by the output gate of C . The *Arithmetic Circuit Identity Testing (ACIT)* problem asks whether the output of a given arithmetic circuit is equal to the zero polynomial.

2.7 The Learning Model

In this paper we work with the *exact learning model* of Angluin (1988): Let f be a *target function*. A *Learner* (*learning algorithm*) may, in each step, propose a hypothesis function h by making an *equivalence query* to a *Teacher*. If h is equivalent to f , then the Teacher returns YES and the Learner succeeds and halts. Otherwise, the Teacher returns NO with a *counterexample*, which is an assignment x such that $h(x) \neq f(x)$. Moreover, the Learner may query the Teacher for the value of the function f on a particular assignment x by making a *membership query* on x . The Teacher returns the value $f(x)$.

We say that a class of functions \mathcal{C} is *exactly learnable* if there is a Learner that for any target function $f \in \mathcal{C}$, outputs a hypothesis $h \in \mathcal{C}$ such that $h(x) = f(x)$ for all assignments x , and does so in time polynomial in the size of a shortest representation of f and the size of a largest counterexample returned by the Teacher. We moreover say that the class \mathcal{C} is *exactly learnable in (randomised) polynomial time* if the learning algorithm can be implemented to run in (randomised) polynomial time in the Turing model.

3. MTA Equivalence is interreducible with ACIT

In this section, we show that the equivalence problem for \mathbb{Q} -multiplicity tree automata is logspace interreducible with **ACIT**. A related result, characterising equivalence of probabilistic visibly pushdown automata on words in terms of polynomial identity testing, was shown by Kiefer et al. (2013). On several occasions in this section, we will implicitly make use of the fact that a composition of two logspace reductions is again a logspace reduction (Arora and Barak, 2009, Lemma 4.17).

3.1 From MTA Equivalence to ACIT

In this section, we present a logspace reduction from the equivalence problem for \mathbb{Q} -MTAs to **ACIT**. We start with the following lemma.

Lemma 8 *Given an integer $n \in \mathbb{N}$ and a \mathbb{Q} -multiplicity tree automaton A over a ranked alphabet Σ , one can compute, in logarithmic space in $|A|$ and n , a variable-free arithmetic circuit that has output $\sum_{t \in T_\Sigma^{<n}} \|A\|(t)$.*

Proof Let $A = (r, \Sigma, \mu, \gamma)$, and let m be the rank of Σ . By definition, it holds that

$$\sum_{t \in T_\Sigma^{<n}} \|A\|(t) = \left(\sum_{t \in T_\Sigma^{<n}} \mu(t) \right) \cdot \gamma. \quad (7)$$

We have $\sum_{t \in T_\Sigma^{<1}} \mu(t) = \sum_{\sigma \in \Sigma_0} \mu(\sigma)$. For every $i \in \mathbb{N}$, it holds that

$$T_\Sigma^{<i+1} = \{\sigma(t_1, \dots, t_k) : k \in \{0, \dots, m\}, \sigma \in \Sigma_k, t_1, \dots, t_k \in T_\Sigma^{<i}\}$$

and thus by bilinearity of Kronecker product,

$$\sum_{t \in T_\Sigma^{<i+1}} \mu(t) = \sum_{k=0}^m \sum_{\sigma \in \Sigma_k} \sum_{t_1 \in T_\Sigma^{<i}} \cdots \sum_{t_k \in T_\Sigma^{<i}} (\mu(t_1) \otimes \cdots \otimes \mu(t_k)) \cdot \mu(\sigma)$$

$$\begin{aligned}
&= \sum_{k=0}^m \sum_{\sigma \in \Sigma_k} \left(\left(\sum_{t_1 \in T_{\Sigma}^{<i}} \mu(t_1) \right) \otimes \cdots \otimes \left(\sum_{t_k \in T_{\Sigma}^{<i}} \mu(t_k) \right) \right) \cdot \mu(\sigma) \\
&= \sum_{k=0}^m \left(\sum_{t \in T_{\Sigma}^{<i}} \mu(t) \right)^{\otimes k} \sum_{\sigma \in \Sigma_k} \mu(\sigma).
\end{aligned} \tag{8}$$

In the following we define a variable-free arithmetic circuit Φ that has output $\sum_{t \in T_{\Sigma}^{<n}} \|A\|(t)$. First, let us denote $G(i) := \sum_{t \in T_{\Sigma}^{<i}} \mu(t)$ for every $i \in \mathbb{N}$. Then by Equation (8) we have $G(i+1) = \sum_{k=0}^m G(i)^{\otimes k} \cdot S(k)$ where $S(k) := \sum_{\sigma \in \Sigma_k} \mu(\sigma)$ for every $k \in \{0, \dots, m\}$. In coordinate notation, for every $j \in [r]$ we have by Equation (1) that

$$G(i+1)_j = \sum_{k=0}^m \sum_{(l_1, \dots, l_k) \in [r]^k} \prod_{a=1}^k G(i)_{l_a} \cdot S(k)_{(l_1, \dots, l_k), j}. \tag{9}$$

We present Φ as a straight-line program, with built-in constants

$$\{\mu_{(l_1, \dots, l_k), j}^{\sigma}, \gamma_j : k \in \{0, \dots, m\}, \sigma \in \Sigma_k, (l_1, \dots, l_k) \in [r]^k, j \in [r]\}$$

representing the entries of the transition matrices and the final weight vector of A , internal variables $\{s_{(l_1, \dots, l_k), j}^k : k \in \{0, \dots, m\}, (l_1, \dots, l_k) \in [r]^k, j \in [r]\}$ and $\{g_{i,j} : i \in [n], j \in [r]\}$ evaluating the entries of matrices $S(k)$ and vectors $G(i)$ respectively, and the final internal variable f computing the value of Φ .

Formally, the straight-line program Φ is given in Table 1. Here the statements are given in indexed-sum and indexed-product notation, which can easily be expanded in terms of the corresponding binary operations. It follows from Equations (7) and (9) that Φ computes $G(n) \cdot \gamma = \sum_{t \in T_{\Sigma}^{<n}} \|A\|(t)$.

The input gates of Φ are labelled with rational numbers. By separately encoding numerators and denominators, we can in logarithmic space reduce Φ to an arithmetic circuit where all input gates are labelled with integers. Moreover, without loss of generality we can assume that every input gate of Φ is labelled with 0 or 1. Any other integer label given in binary can be encoded as an arithmetic circuit.

Recalling that a composition of two logspace reductions is again a logspace reduction, we conclude that the entire computation takes logarithmic space in $|A|$ and n . \blacksquare

Before presenting the reduction in Proposition 10, we recall the following characterisation (Seidl, 1990, Theorem 4.2) of equivalence of two multiplicity tree automata over an arbitrary field.

Proposition 9 (Seidl, 1990) *Suppose A and B are multiplicity tree automata of dimension n_1 and n_2 , respectively, and over a ranked alphabet Σ . Then, A and B are equivalent if and only if $\|A\|(t) = \|B\|(t)$ for every $t \in T_{\Sigma}^{<n_1+n_2}$.*

Proposition 10 *The equivalence problem for \mathbb{Q} -multiplicity tree automata is logspace reducible to **ACIT**.*

-
1. For $j \in [r]$ do $g_{1,j} \leftarrow \sum_{\sigma \in \Sigma_0} \mu_j^\sigma$
 2. For $k \in \{0, \dots, m\}$, $(l_1, \dots, l_k) \in [r]^k$, $j \in [r]$ do $s_{(l_1, \dots, l_k), j}^k \leftarrow \sum_{\sigma \in \Sigma_k} \mu_{(l_1, \dots, l_k), j}^\sigma$
 3. For $i = 1$ to $n - 1$ do
 - 3.1. For $k \in \{0, \dots, m\}$, $(l_1, \dots, l_k) \in [r]^k$, $j \in [r]$ do

$$h_{(l_1, \dots, l_k), j}^{i,k} \leftarrow \prod_{a=1}^k g_{i, l_a} \cdot s_{(l_1, \dots, l_k), j}^k$$
 - 3.2. For $j \in [r]$ do

$$g_{i+1, j} \leftarrow \sum_{k=0}^m \sum_{(l_1, \dots, l_k) \in [r]^k} h_{(l_1, \dots, l_k), j}^{i,k}$$
 4. For $j \in [r]$ do $f_j \leftarrow g_{n,j} \cdot \gamma_j$
 5. $f \leftarrow \sum_{j \in [r]} f_j$.
-

Table 1: Straight-line program Φ

Proof Let A and B be \mathbb{Q} -multiplicity tree automata over a ranked alphabet Σ , and let n be the sum of their dimensions. Proposition 2 implies that

$$\begin{aligned} \sum_{t \in T_\Sigma^{\leq n}} (\|A\|(t) - \|B\|(t))^2 &= \sum_{t \in T_\Sigma^{\leq n}} (\|A\|(t)^2 + \|B\|(t)^2 - 2\|A\|(t)\|B\|(t)) \\ &= \sum_{t \in T_\Sigma^{\leq n}} (\|A \times A\|(t) + \|B \times B\|(t) - 2\|A \times B\|(t)). \end{aligned}$$

Thus by Proposition 9, automata A and B are equivalent if and only if

$$\sum_{t \in T_\Sigma^{\leq n}} \|A \times A\|(t) + \sum_{t \in T_\Sigma^{\leq n}} \|B \times B\|(t) - 2 \sum_{t \in T_\Sigma^{\leq n}} \|A \times B\|(t) = 0. \quad (10)$$

We know from Proposition 2 that automata $A \times A$, $B \times B$, and $A \times B$ can be computed in logarithmic space. Thus by Lemma 8 one can compute, in logarithmic space in $|A|$ and $|B|$, variable-free arithmetic circuits that have outputs $\sum_{t \in T_\Sigma^{\leq n}} \|A \times A\|(t)$, $\sum_{t \in T_\Sigma^{\leq n}} \|B \times B\|(t)$, and $\sum_{t \in T_\Sigma^{\leq n}} \|A \times B\|(t)$ respectively. Using Equation (10), we can now easily construct a variable-free arithmetic circuit that has output 0 if and only if A and B are equivalent. ■

3.2 From ACIT to MTA Equivalence

We now present a converse reduction: from **ACIT** to the equivalence problem for \mathbb{Q} -MTAs.

Allender et al. (2009, Proposition 2.2) give a logspace reduction of the general **ACIT** problem to the special case of **ACIT** for variable-free circuits. The latter can, by representing arbitrary integers as differences of two non-negative integers, be reformulated as the problem of deciding whether two variable-free arithmetic circuits with only $+$ and \times -internal gates compute the same number.

Proposition 11 *ACIT is logspace reducible to the equivalence problem for \mathbb{Q} -multiplicity tree automata.*

Proof Let C_1 and C_2 be two variable-free arithmetic circuits whose internal gates are labelled with $+$ or \times . By padding with extra gates, without loss of generality we can assume that in each circuit the children of a height- i gate both have height $i - 1$, $+$ -gates have even height, \times -gates have odd height, and the output gate has an even height h .

In the following we define two \mathbb{Q} -MTAs, A_1 and A_2 , that are equivalent if and only if circuits C_1 and C_2 have the same output. Automata A_1 and A_2 are both defined over a ranked alphabet $\Sigma = \{\sigma_0, \sigma_1, \sigma_2\}$ where σ_0 is a nullary, σ_1 a unary, and σ_2 a binary symbol. Intuitively, automata A_1 and A_2 both recognise the common ‘tree-unfolding’ of C_1 and C_2 .

We now derive A_1 from C_1 ; A_2 is analogously derived from C_2 . Let $\{v_1, \dots, v_r\}$ be the set of gates of C_1 where v_r is the output gate. Automaton A_1 has a state q_i for every gate v_i of C_1 . Formally, $A_1 = (r, \Sigma, \mu, e_r)$ where for every $i \in [r]$:

- If v_i is an input gate with label 1 then $\mu(\sigma_0)_i = 1$, otherwise $\mu(\sigma_0)_i = 0$.
- If v_i is a $+$ -gate with children v_{j_1} and v_{j_2} then $\mu(\sigma_1)_{j_1,i} = \mu(\sigma_1)_{j_2,i} = 1$ if $j_1 \neq j_2$, $\mu(\sigma_1)_{j_1,i} = 2$ if $j_1 = j_2$, and $\mu(\sigma_1)_{l,i} = 0$ for every $l \notin \{j_1, j_2\}$. If v_i is an input gate or a \times -gate then $\mu(\sigma_1)^i = 0_{r \times 1}$.
- If v_i is a \times -gate with children v_{j_1} and v_{j_2} then $\mu(\sigma_2)_{(j_1,j_2),i} = 1$, and $\mu(\sigma_2)_{(l_1,l_2),i} = 0$ for every $(l_1, l_2) \neq (j_1, j_2)$. If v_i is an input gate or a $+$ -gate then $\mu(\sigma_2)^i = 0_{r^2 \times 1}$.

Define a sequence of trees $(t_n)_{n \in \mathbb{N}_0} \subseteq T_\Sigma$ by $t_0 = \sigma_0$, $t_{n+1} = \sigma_1(t_n)$ for n odd, and $t_{n+1} = \sigma_2(t_n, t_n)$ for n even. In the following, we show that $\|A_1\|(t_h) = f_{C_1}$. For every gate v of C_1 , by assumption it holds that all paths from v to the output gate have equal length. We now prove that for every $i \in [r]$,

$$\mu(t_{h_i})_i = f_{v_i} \tag{11}$$

where $h_i := \text{height}(v_i)$. We use induction on $h_i \in \{0, \dots, h\}$. For the base case, let $h_i = 0$. Then, v_i is an input gate and thus by definition of automaton A_1 we have

$$\mu(t_{h_i})_i = \mu(t_0)_i = \mu(\sigma_0)_i = f_{v_i}.$$

For the induction step, let $n \in [h]$ and assume that Equation (11) holds for every gate v_i of height less than n . Take an arbitrary gate v_i of C_1 such that $h_i = n$. Let gates v_{j_1} and v_{j_2} be the children of v_i . Then $h_{j_1} = h_{j_2} = h_i - 1 = n - 1$ by assumption. The induction hypothesis now implies that $\mu(t_{h_{j_1}-1})_{j_1} = f_{v_{j_1}}$ and $\mu(t_{h_{j_2}-1})_{j_2} = f_{v_{j_2}}$. Depending on the label of v_i , there are two possible cases as follows:

(i) If v_i is a $+$ -gate, then h_i is even and thus by definition of A_1 we have

$$\begin{aligned}\mu(t_{h_i})_i &= \mu(\sigma_1(t_{h_i-1}))_i = \mu(t_{h_i-1}) \cdot \mu(\sigma_1)^i \\ &= \mu(t_{h_i-1})_{j_1} + \mu(t_{h_i-1})_{j_2} = f_{v_{j_1}} + f_{v_{j_2}} = f_{v_i}.\end{aligned}$$

(ii) If v_i is a \times -gate, then h_i is odd and thus by definition of A_1 and Equation (1) we have

$$\begin{aligned}\mu(t_{h_i})_i &= \mu(\sigma_2(t_{h_i-1}, t_{h_i-1}))_i = \mu(t_{h_i-1})^{\otimes 2} \cdot \mu(\sigma_2)^i \\ &= \mu(t_{h_i-1})_{j_1} \cdot \mu(t_{h_i-1})_{j_2} = f_{v_{j_1}} \cdot f_{v_{j_2}} = f_{v_i}.\end{aligned}$$

This completes the proof of Equation (11) by induction. Now for the output gate v_r of C_1 , we get from Equation (11) that $\mu(t_h)_r = f_{v_r}$ since $h_r = h$. Therefore,

$$\|A_1\|(t_h) = \mu(t_h) \cdot e_r = \mu(t_h)_r = f_{v_r} = f_{C_1}.$$

Analogously, it holds that $\|A_2\|(t_h) = f_{C_2}$. It is moreover clear by construction that $\|A_1\|(t) = 0$ and $\|A_2\|(t) = 0$ for every $t \in T_\Sigma \setminus \{t_h\}$. Therefore, automata A_1 and A_2 are equivalent if and only if arithmetic circuits C_1 and C_2 have the same output. \blacksquare

4. The Learning Algorithm

In this section, we give an exact learning algorithm for multiplicity tree automata. Our algorithm is polynomial in the size of a minimal automaton equivalent to the target and the size of a largest counterexample given as a DAG. As seen in Example 1, DAG counterexamples can be exponentially more succinct than tree counterexamples. Therefore, achieving a polynomial bound in the context of DAG representations is a more exacting criterion.

Over an arbitrary field \mathbb{F} , the algorithm can be seen as running on a Blum-Shub-Smale machine that can write and read field elements to and from its memory at unit cost and that can also perform arithmetic operations and equality tests on field elements at unit cost (see Arora and Barak, 2009). Over \mathbb{Q} , the algorithm can be implemented in randomised polynomial time by representing rationals as arithmetic circuits and using a coRP algorithm for equality testing of such circuits (see Allender et al., 2009).

This section is organised as follows: In Section 4.1 we present the algorithm. In Section 4.2 we prove correctness on trees, and then argue in Section 4.3 that the algorithm can be faithfully implemented using a DAG representation of trees. Finally, in Section 4.4 we give a complexity analysis of the algorithm assuming the DAG representation.

4.1 The Algorithm

Let $f \in \text{Rec}(\Sigma, \mathbb{F})$ be the target function. The algorithm learns an MTA-representation of f using its Hankel matrix H , which has finite rank over \mathbb{F} by Theorem 3. At each stage, the algorithm maintains the following data:

- An integer $n \in \mathbb{N}$.
- A set of n ‘rows’ $X = \{t_1, \dots, t_n\} \subseteq T_\Sigma$.

- A finite set of ‘columns’ $Y \subseteq C_\Sigma$, where $\square \in Y$.
- A submatrix $H_{X,Y}$ of H that has full row rank.

These data determine a *hypothesis automaton* A of dimension n , whose states correspond to the rows of $H_{X,Y}$, with the i^{th} row being the state reached after reading the tree t_i . The Learner makes an equivalence query on hypothesis A . In case the Teacher answers NO, the Learner receives a counterexample z . The Learner then parses z bottom-up to find a minimal subtree of z that is also a counterexample, and uses that subtree to augment the sets of rows and columns.

Formally, the algorithm LMTA is given in Table 2. Here for any k -ary symbol $\sigma \in \Sigma$ we define $\sigma(X, \dots, X) := \{\sigma(t_{i_1}, \dots, t_{i_k}) : (i_1, \dots, i_k) \in [n]^k\}$.

4.2 Correctness Proof

In this section, we prove the correctness of the exact learning algorithm LMTA. Specifically, we show that, given a target $f \in \text{Rec}(\Sigma, \mathbb{F})$, algorithm LMTA outputs a minimal MTA-representation of f after at most $\text{rank}(H)$ iterations of the main loop.

The correctness proof naturally breaks down into several lemmas. First, we show that matrix $H_{X,Y}$ has full row rank.

Lemma 12 *Linear independence of the set of vectors $\{H_{t_1,Y}, \dots, H_{t_n,Y}\}$ is an invariant of the loop consisting of Step 2 and Step 3.*

Proof We argue inductively on the number of iterations of the loop. The base case $n = 1$ clearly holds since $f(z) \neq 0$.

For the induction step, suppose that the set $\{H_{t_1,Y}, \dots, H_{t_n,Y}\}$ is linearly independent at the start of an iteration of the loop. If a tree $t \in T_\Sigma$ is added to X during Step 2.1, then $H_{t,Y}$ is not a linear combination of $H_{t_1,Y}, \dots, H_{t_n,Y}$, and therefore $H_{t_1,Y}, \dots, H_{t_n,Y}, H_{t,Y}$ are linearly independent vectors. Hence, set $\{H_{t_1,Y}, \dots, H_{t_n,Y}\}$ is linearly independent at the start of Step 3.

Unless the algorithm halts in Step 3.1, it proceeds to Step 3.2 where the set of columns Y is increased, which clearly preserves linear independence of vectors $H_{t_1,Y}, \dots, H_{t_n,Y}$. If a tree τ_j is added to X in Step 3.3, then $H_{\tau_j,Y}$ is not a linear combination of $H_{t_1,Y}, \dots, H_{t_n,Y}$ which implies that the vectors $H_{t_1,Y}, \dots, H_{t_n,Y}, H_{\tau_j,Y}$ are linearly independent. Hence, the set $\{H_{t_1,Y}, \dots, H_{t_n,Y}\}$ is linearly independent at the start of the next iteration of the loop. ■

Secondly, we show that Step 2.2 of LMTA can always be performed.

Lemma 13 *Whenever Step 2.2 starts, for every $k \in \{0, \dots, m\}$ and $\sigma \in \Sigma_k$ there exists a unique matrix $\mu(\sigma) \in \mathbb{F}^{n^k \times n}$ satisfying Equation (12).*

Proof Take any $(i_1, \dots, i_k) \in [n]^k$. Step 2.1 ensures that $H_{\sigma(t_{i_1}, \dots, t_{i_k}), Y}$ can be represented as a linear combination of vectors $H_{t_1,Y}, \dots, H_{t_n,Y}$. This representation is unique since $H_{t_1,Y}, \dots, H_{t_n,Y}$ are linearly independent vectors by Lemma 12. Row $\mu(\sigma)_{(i_1, \dots, i_k)} \in \mathbb{F}^{1 \times n}$ is therefore uniquely defined by the equation $\mu(\sigma)_{(i_1, \dots, i_k)} \cdot H_{X,Y} = H_{\sigma(t_{i_1}, \dots, t_{i_k}), Y}$. ■

Algorithm LMTA

Target: $f \in \text{Rec}(\Sigma, \mathbb{F})$, where Σ has rank m and \mathbb{F} is a field

1. Make an equivalence query on the 0-dimensional \mathbb{F} -MTA over Σ .
 If the answer is YES then **output** the 0-dimensional \mathbb{F} -MTA over Σ and halt.
 Otherwise the answer is NO and z is a counterexample. Initialise:
 $n \leftarrow 1, t_n \leftarrow z, X \leftarrow \{t_n\}, Y \leftarrow \{\square\}$.
 2. 2.1. For every $k \in \{0, \dots, m\}$, $\sigma \in \Sigma_k$, and $(i_1, \dots, i_k) \in [n]^k$:
 If $H_{\sigma(t_{i_1}, \dots, t_{i_k}), Y}$ is not a linear combination of $H_{t_1, Y}, \dots, H_{t_n, Y}$ then
 $n \leftarrow n + 1, t_n \leftarrow \sigma(t_{i_1}, \dots, t_{i_k}), X \leftarrow X \cup \{t_n\}$.
 - 2.2. Define an \mathbb{F} -MTA $A = (n, \Sigma, \mu, \gamma)$ as follows:
 - $\gamma = H_{X, \square}$.
 - For every $k \in \{0, \dots, m\}$ and $\sigma \in \Sigma_k$:
 Define matrix $\mu(\sigma) \in \mathbb{F}^{n^k \times n}$ by the equation

$$\mu(\sigma) \cdot H_{X, Y} = H_{\sigma(X, \dots, X), Y}. \quad (12)$$
 3. 3.1. Make an equivalence query on A .
 If the answer is YES then **output** A and halt.
 Otherwise the answer is NO and z is a counterexample. Searching bottom-up,
 find a subtree $\sigma(\tau_1, \dots, \tau_k)$ of z that satisfies the following two conditions:
 - (i) For every $j \in [k]$, $H_{\tau_j, Y} = \mu(\tau_j) \cdot H_{X, Y}$.
 - (ii) For some $c \in Y$, $H_{\sigma(\tau_1, \dots, \tau_k), c} \neq \mu(\sigma(\tau_1, \dots, \tau_k)) \cdot H_{X, c}$.
 - 3.2. For every $j \in [k]$ and $(i_1, \dots, i_{j-1}) \in [n]^{j-1}$:
 $Y \leftarrow Y \cup \{c[\sigma(t_{i_1}, \dots, t_{i_{j-1}}, \square, \tau_{j+1}, \dots, \tau_k)]\}$.
 - 3.3. For every $j \in [k]$:
 If $H_{\tau_j, Y}$ is not a linear combination of $H_{t_1, Y}, \dots, H_{t_n, Y}$ then
 $n \leftarrow n + 1, t_n \leftarrow \tau_j, X \leftarrow X \cup \{t_n\}$.
 - 3.4. Go to 2.
-

Table 2: Exact learning algorithm LMTA for the class of multiplicity tree automata

Thirdly, we show that Step 3.1 of LMTA can always be performed.

Lemma 14 *Suppose that upon making an equivalence query on A in Step 3.1, the Learner receives the answer NO and a counterexample z . Then there exists a subtree $\sigma(\tau_1, \dots, \tau_k)$ of z , where $k \in \{0, \dots, m\}$, $\sigma \in \Sigma_k$, and $\tau_1, \dots, \tau_k \in T_\Sigma$, that satisfies the following two conditions:*

- (i) *For every $j \in [k]$, $H_{\tau_j, Y} = \mu(\tau_j) \cdot H_{X, Y}$.*
- (ii) *For some $c \in Y$, $H_{\sigma(\tau_1, \dots, \tau_k), c} \neq \mu(\sigma(\tau_1, \dots, \tau_k)) \cdot H_{X, c}$.*

Proof Towards a contradiction, assume that there is no subtree $\sigma(\tau_1, \dots, \tau_k)$ of z satisfying conditions (i) and (ii). We claim that then for every subtree τ of z , it holds that

$$H_{\tau,Y} = \mu(\tau) \cdot H_{X,Y}. \quad (13)$$

In the following we prove this claim using induction on $\text{height}(\tau)$. The base case $\tau \in \Sigma_0$ follows immediately from Equation (12). For the induction step, let $0 \leq h < \text{height}(z)$ and assume that Equation (13) holds for every subtree $\tau \in T_{\Sigma}^{\leq h}$ of z . Take an arbitrary subtree $\tau \in T_{\Sigma}^{h+1}$ of z . Then $\tau = \sigma(\tau_1, \dots, \tau_k)$ for some $k \in [m]$, $\sigma \in \Sigma_k$, and $\tau_1, \dots, \tau_k \in T_{\Sigma}^{\leq h}$, where τ_1, \dots, τ_k are subtrees of z . The induction hypothesis implies that $H_{\tau_j,Y} = \mu(\tau_j) \cdot H_{X,Y}$ holds for every $j \in [k]$. Hence, τ satisfies condition (i). By assumption, no subtree of z satisfies both conditions (i) and (ii). Thus τ does not satisfy condition (ii), i.e., it holds that $H_{\tau,Y} = \mu(\tau) \cdot H_{X,Y}$. This completes the proof by induction.

Equation (13) for $\tau = z$ gives $H_{z,Y} = \mu(z) \cdot H_{X,Y}$. Since $\square \in Y$, this in particular implies that

$$f(z) = H_{z,\square} = \mu(z) \cdot H_{X,\square} = \mu(z) \cdot \gamma = \|A\|(z),$$

which yields a contradiction since z is a counterexample for the hypothesis A . ■

Finally, we show that the row set X grows by at least 1 in each iteration of the main loop.

Lemma 15 *Every complete iteration of the Step 2 - 3 loop strictly increases the cardinality of X .*

Proof It suffices to show that in Step 3.3 at least one of the trees τ_1, \dots, τ_k is added to X . By Lemma 12, at the start of Step 3.2 vectors $H_{t_1,Y}, \dots, H_{t_n,Y}$ are linearly independent. Thus by condition (i) of Step 3.1, for every $j \in [k]$ it holds that

$$H_{\tau_j,Y} = \mu(\tau_j) \cdot H_{X,Y} \quad (14)$$

and, moreover, Equation (14) is the unique representation of vector $H_{\tau_j,Y}$ as a linear combination of $H_{t_1,Y}, \dots, H_{t_n,Y}$. Clearly, vectors $H_{t_1,Y}, \dots, H_{t_n,Y}$ remain linearly independent when Step 3.2 ends.

Towards a contradiction, assume that in Step 3.3 none of the trees τ_1, \dots, τ_k were added to X . This means that for every $j \in [k]$, vector $H_{\tau_j,Y}$ can be represented as a linear combination of $H_{t_1,Y}, \dots, H_{t_n,Y}$. The latter representation is unique, since vectors $H_{t_1,Y}, \dots, H_{t_n,Y}$ are linearly independent, and is given by Equation (14). By condition (ii) of Step 3.1 and Equations (12) and (1), we now have that

$$\begin{aligned} H_{\sigma(\tau_1, \dots, \tau_k), c} &\neq \mu(\sigma(\tau_1, \dots, \tau_k)) \cdot H_{X,c} \\ &= (\mu(\tau_1) \otimes \dots \otimes \mu(\tau_k)) \cdot \mu(\sigma) \cdot H_{X,c} \\ &= (\mu(\tau_1) \otimes \dots \otimes \mu(\tau_k)) \cdot H_{\sigma(X, \dots, X), c} \\ &= \sum_{(i_1, \dots, i_k) \in [n]^k} \left(\prod_{j=1}^k \mu(\tau_j)_{i_j} \right) \cdot H_{\sigma(t_{i_1}, \dots, t_{i_k}), c}. \end{aligned} \quad (15)$$

By Step 3.2, we have that $c[\sigma(t_{i_1}, \dots, t_{i_{j-1}}, \square, \tau_{j+1}, \dots, \tau_k)] \in Y$ for every $j \in [k]$ and $(i_1, \dots, i_{j-1}) \in [n]^{j-1}$. Thus by Equation (14) for $j = k$, we have

$$\begin{aligned}
& \sum_{(i_1, \dots, i_k) \in [n]^k} \left(\prod_{j=1}^k \mu(\tau_j)_{i_j} \right) \cdot H_{\sigma(t_{i_1}, \dots, t_{i_k}), c} \\
&= \sum_{(i_1, \dots, i_{k-1}) \in [n]^{k-1}} \left(\prod_{j=1}^{k-1} \mu(\tau_j)_{i_j} \right) \sum_{i \in [n]} \mu(\tau_k)_i \cdot H_{t_i, c[\sigma(t_{i_1}, \dots, t_{i_{k-1}}, \square)]} \\
&= \sum_{(i_1, \dots, i_{k-1}) \in [n]^{k-1}} \left(\prod_{j=1}^{k-1} \mu(\tau_j)_{i_j} \right) \cdot (\mu(\tau_k) \cdot H_{X, c[\sigma(t_{i_1}, \dots, t_{i_{k-1}}, \square)]}) \\
&= \sum_{(i_1, \dots, i_{k-1}) \in [n]^{k-1}} \left(\prod_{j=1}^{k-1} \mu(\tau_j)_{i_j} \right) \cdot H_{\tau_k, c[\sigma(t_{i_1}, \dots, t_{i_{k-1}}, \square)]}. \tag{16}
\end{aligned}$$

Proceeding inductively as above and applying Equation (14) for every $j \in \{k-1, \dots, 1\}$, we get that the expression of (16) is equal to $H_{\tau_1, c[\sigma(\square, \tau_2, \dots, \tau_k)]}$. However, this contradicts Equation (15). The result follows. \blacksquare

Putting together Lemmas 12 - 15, we conclude the following.

Proposition 16 *Let Σ be a ranked alphabet and \mathbb{F} be a field. Let $f \in \text{Rec}(\Sigma, \mathbb{F})$, let H be the Hankel matrix of f , and r be the rank (over \mathbb{F}) of H . On target f , the algorithm LMTA outputs a minimal MTA-representation of f after at most r iterations of the loop consisting of Step 2 and Step 3.*

Proof Lemmas 13 and 14 show that every step of the algorithm LMTA can be performed.

Theorem 3 implies that r is finite. From Lemma 12 we know that whenever Step 2 starts, $H_{X,Y}$ is a full row rank matrix and thus $n \leq r$. Lemma 15 implies that n increases by at least 1 in each iteration of the Step 2 - 3 loop. Therefore, the number of iterations of the loop is at most r .

The proof follows by observing that LMTA halts only upon receiving the answer YES to an equivalence query. \blacksquare

4.3 Succinct Representations

We now explain how algorithm LMTA can be correctly implemented using a DAG representation of trees. In particular, we assume that membership queries are made on Σ -DAGs, that the counterexamples are given as Σ -DAGs, the elements of X are Σ -DAGs, and the elements of Y are DAG representations of Σ -contexts, i.e., $(\{\square\} \cup \Sigma)$ -DAGs.

As shown in Section 2.5, multiplicity tree automata can run directly on DAGs and, moreover, they assign equal weight to a DAG and to its tree unfolding. Crucially also, as explained in the proof of Theorem 17, Step 3.1 can be run directly on a DAG representation

of the counterexample, without unfolding. In particular, Step 3.1 involves multiple executions of the hypothesis automaton on trees. By Proposition 7, we can faithfully carry out this step using DAG representations of trees. Step 3.1 also involves considering all subtrees of a given counterexample. However, by Proposition 5 this is equivalent to looking at all sub-DAGs of a DAG representation of the counterexample. At various points in the algorithm, we take $c \in Y$, $t \in X$ and compute their concatenation $c[t]$ in order to determine the corresponding entry $H_{t,c}$ of the Hankel matrix by a membership query. Proposition 6 implies that this can be done faithfully using DAG representations of Σ -trees and Σ -contexts.

4.4 Complexity Analysis

In this section we give a complexity analysis of our algorithm, and compare it to the best previously-known exact learning algorithm for multiplicity tree automata (Habrard and Oncina, 2006) showing in particular an exponential improvement on the query complexity and the running time in the worst case.

Theorem 17 *Let $f \in \text{Rec}(\Sigma, \mathbb{F})$ where Σ has rank m and \mathbb{F} is a field. Let A be a minimal MTA-representation of f , and let r be the dimension of A . Then, f is learnable by the algorithm LMTA, making $r + 1$ equivalence queries, $|A|^2 + |A| \cdot s$ membership queries, and $O(|A|^2 + |A| \cdot r \cdot s)$ arithmetic operations, where s denotes the size of a largest counterexample z , represented as a DAG, that is obtained during the execution of the algorithm.*

Proof Proposition 16 implies that, on target f , algorithm LMTA outputs a minimal MTA-representation of f after at most r iterations of the Step 2 - 3 loop, thereby making at most $r + 1$ equivalence queries.

Let H be the Hankel matrix of f . From Lemma 12 we know that matrix $H_{X,Y}$ has full row rank, which implies that $|X| \leq r$. As for the cardinality of the column set Y , at the end of Step 1 we have $|Y| = 1$. Furthermore, in each iteration of Step 3.2 the number of columns added to Y is at most

$$\sum_{j=1}^k n^{j-1} \leq \sum_{j=1}^k r^{j-1} = \frac{r^k - 1}{r - 1} \leq \frac{r^m - 1}{r - 1},$$

where k and n are as defined in Step 3.2. Since the number of iterations of Step 3.2 is at most $r - 1$, we have $|Y| \leq r^m$.

The number of membership queries made in Step 2 over the whole algorithm is

$$\left(\sum_{\sigma \in \Sigma} |\sigma(X, \dots, X)| + |X| \right) \cdot |Y|$$

because the Learner needs to ask for the values of the entries of matrices $H_{X,Y}$ and $H_{\sigma(X, \dots, X), Y}$ for every $\sigma \in \Sigma$.

To analyse the number of membership queries made in Step 3, we now detail the procedure by which an appropriate sub-DAG of the counterexample z is found in Step 3.1. By Lemma 14, there exists a sub-DAG τ of z such that $H_{\tau,Y} \neq \mu(\tau) \cdot H_{X,Y}$. Thus given a counterexample z in Step 3.1, the procedure for finding a required sub-DAG of z is as

follows: Check if $H_{\tau,Y} = \mu(\tau) \cdot H_{X,Y}$ for every sub-DAG τ of z in a nondecreasing order of height; stop when a sub-DAG τ is found such that $H_{\tau,Y} \neq \mu(\tau) \cdot H_{X,Y}$.

In each iteration of Step 3, the Learner makes $\text{size}(z) \cdot |Y| \leq s \cdot |Y|$ membership queries because, for every sub-DAG τ of z , the Learner needs to ask for the values of the entries of vector $H_{\tau,Y}$. All together, the number of membership queries made during the execution of the algorithm is at most

$$\begin{aligned} & \left(\sum_{\sigma \in \Sigma} |\sigma(X, \dots, X)| + |X| \right) \cdot |Y| + (r-1) \cdot s \cdot |Y| \\ & \leq \left(\sum_{\sigma \in \Sigma} r^{rk(\sigma)} + r \right) \cdot r^m + (r-1) \cdot s \cdot r^m \leq |A|^2 + |A| \cdot s. \end{aligned}$$

As for the arithmetic complexity, in Step 2.1 one can determine if a vector $H_{\sigma(t_{i_1}, \dots, t_{i_k}), Y}$ is a linear combination of $H_{t_1, Y}, \dots, H_{t_n, Y}$ via Gaussian elimination using $O(n^2 \cdot |Y|)$ arithmetic operations (see Cohen, 1993, Section 2.3). Analogously, in Step 3.3 one can determine if $H_{\tau_j, Y}$ is a linear combination of $H_{t_1, Y}, \dots, H_{t_n, Y}$ via Gaussian elimination using $O(n^2 \cdot |Y|)$ arithmetic operations. Since $|X| \leq r$ and $|Y| \leq r^m$, all together Step 2.1 and Step 3.3 require at most $O(|A|^2)$ arithmetic operations.

Lemma 13 implies that in each iteration of Step 2.2, for any $\sigma \in \Sigma$ there exists a unique matrix $\mu(\sigma) \in \mathbb{F}^{n^{rk(\sigma)} \times n}$ that satisfies Equation (12). To perform an iteration of Step 2.2, we first put matrix $H_{X,Y}$ in echelon form and then, for each $\sigma \in \Sigma$, solve Equation (12) for $\mu(\sigma)$ by back substitution. It follows from standard complexity bounds on the conversion of matrices to echelon form (Cohen, 1993, Section 2.3) that the total operation count for Step 2.2 can be bounded above by $O(|A|^2)$.

Finally, we consider the arithmetic complexity of Step 3.1. In every iteration, for each sub-DAG τ of the counterexample z the Learner needs to compute the vector $\mu(\tau)$ and the product $\mu(\tau) \cdot H_{X,Y}$. Note that $\mu(\tau)$ can be computed bottom-up from the sub-DAGs of τ . Since z has at most s sub-DAGs, Step 3.1 requires at most $O(|A| \cdot r \cdot s)$ arithmetic operations. All together, the algorithm requires at most $O(|A|^2 + |A| \cdot r \cdot s)$ arithmetic operations. ■

Algorithm LMTA can be used to show that over \mathbb{Q} , multiplicity tree automata are exactly learnable in randomised polynomial time. The key idea is to represent numbers as arithmetic circuits. In executing LMTA, the Learner need only perform arithmetic operations on circuits (addition, subtraction, multiplication, and division), which can be done in constant time, and equality testing, which can be done in coRP (see Arora and Barak, 2009). These suffice for all the operations detailed in the proof of Theorem 17; in particular they suffice for Gaussian elimination, which can be used to implement the linear independence checks in LMTA.

The complexity of algorithm LMTA should be compared to the complexity of the algorithm of Habrard and Oncina (2006), which learns multiplicity tree automata by making $r+1$ equivalence queries, $|A| \cdot s$ membership queries, and a number of arithmetic operations polynomial in $|A|$ and s , where s is the size of the largest counterexample given as a tree. Note that the algorithm of Habrard and Oncina (2006) cannot be straightforwardly adapted to work directly with DAG representations of trees since when given a counterexample z ,

every suffix of z is added to the set of columns. However, the tree unfolding of a DAG can have exponentially many different suffixes in the size of the DAG. For example, the DAG in Figure 2 has size n , and its tree unfolding, shown in Figure 1, has $O(2^n)$ different suffixes.

5. Lower Bounds on Query Complexity of Learning MTA

In this section, we study lower bounds on the query complexity of learning multiplicity tree automata in the exact learning model. Our results generalise the corresponding lower bounds for learning multiplicity word automata by Bisht et al. (2006), and make no assumption about the computational model of the learning algorithm.

First, we give a lower bound on the query complexity of learning multiplicity tree automata over an arbitrary field, which is the situation of our algorithm in Section 4.

Theorem 18 *Any exact learning algorithm that learns the class of multiplicity tree automata of dimension at most r , over a ranked alphabet (Σ, rk) and any field, must make at least $\sum_{\sigma \in \Sigma} r^{rk(\sigma)+1} - r^2$ queries.*

Proof Take an arbitrary exact learning algorithm L that learns the class of multiplicity tree automata of dimension at most r , over a ranked alphabet (Σ, rk) and over any field. Let \mathbb{F} be any field.

Let $\mathbb{K} := \mathbb{F}(\{z_{i,j}^\sigma : \sigma \in \Sigma, i \in [r^{rk(\sigma)}], j \in [r]\})$ be an extension field of \mathbb{F} , where the set $\{z_{i,j}^\sigma : \sigma \in \Sigma, i \in [r^{rk(\sigma)}], j \in [r]\}$ is algebraically independent over \mathbb{F} . Let us define a ‘generic’ \mathbb{K} -multiplicity tree automaton $A := (r, \Sigma, \mu, \gamma)$ where $\gamma = e_1 \in \mathbb{F}^{r \times 1}$ and $\mu(\sigma) = [z_{i,j}^\sigma]_{i,j} \in \mathbb{K}^{r^{rk(\sigma)} \times r}$ for every $\sigma \in \Sigma$. Define a tree series $f := \|A\|$. Since every \mathbb{F} -multiplicity tree automaton can be obtained from A by substituting values from \mathbb{F} for the variables $z_{i,j}^\sigma$, if the Hankel matrix of f had rank less than r then every r -dimensional \mathbb{F} -multiplicity tree automaton would have Hankel matrix of rank less than r . Therefore, the Hankel matrix of f has rank r .

We run algorithm L on the target function f . By assumption, the output of L is an MTA $A' = (r, \Sigma, \mu', \gamma')$ such that $\|A'\| \equiv f$. Let n be the number of queries made by L on target f . Let $t_1, \dots, t_n \in T_\Sigma$ be the trees on which L either made a membership query, or which were received as counterexample to an equivalence query. Then for every $l \in [n]$, there exists a multivariate polynomial $p_l \in \mathbb{F}[(z_{i,j}^\sigma)_{i,j,\sigma}]$ such that $f(t_l) = p_l$.

Note that A and A' are both minimal MTA-representations of f . Thus by Theorem 4, there exists an invertible matrix $U \in \mathbb{K}^{r \times r}$ such that $\gamma = U \cdot \gamma'$ and $\mu(\sigma) = U^{\otimes rk(\sigma)} \cdot \mu'(\sigma) \cdot U^{-1}$ for every $\sigma \in \Sigma$. This implies that the entries of matrices $\mu(\sigma)$, $\sigma \in \Sigma$, lie in an extension of \mathbb{F} generated by the entries of U and $\{p_l : l \in [n]\}$. Since the entries of matrices $\mu(\sigma)$, $\sigma \in \Sigma$, form an algebraically independent set over \mathbb{F} , their number is at most $r^2 + n$. ■

One may wonder whether a learning algorithm could do better over a fixed field \mathbb{F} by exploiting particular features of the field. In this setting, we have the following lower bound.

Theorem 19 *Let \mathbb{F} be an arbitrary field. Any exact learning algorithm that learns the class of \mathbb{F} -multiplicity tree automata of dimension at most r , over a ranked alphabet (Σ, rk) with*

at least one unary symbol and rank m , must make number of queries at least

$$\frac{1}{2^{m+1}} \cdot \left(\sum_{\sigma \in \Sigma} r^{rk(\sigma)+1} - r^2 - r \right).$$

Proof Without loss of generality, we can assume that r is even and define $n := r/2$. Let L be an exact learning algorithm for the class of \mathbb{F} -multiplicity tree automata of dimension at most r , over a ranked alphabet (Σ, rk) with $rk^{-1}(\{1\}) \neq \emptyset$. We will identify a class of functions \mathcal{C} such that L has to make at least $\sum_{\sigma \in \Sigma} n^{rk(\sigma)+1} - n^2 - n$ queries to distinguish between the members of \mathcal{C} .

Let $\sigma_0, \sigma_1 \in \Sigma$ be nullary and unary symbols respectively. Let $P \in \mathbb{F}^{n \times n}$ be the permutation matrix corresponding to the cycle $(1, 2, \dots, n)$. Define \mathcal{A} to be the set of all \mathbb{F} -multiplicity tree automata $(2n, \Sigma, \mu, \gamma)$ where

- $\mu(\sigma_0) = \begin{bmatrix} 1 & 0 \end{bmatrix} \otimes e_1$ and $\mu(\sigma_1) = I_2 \otimes P$;
- For each k -ary symbol $\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}$, there exists $B(\sigma) \in \mathbb{F}^{n^k \times n}$ such that

$$\mu(\sigma) = \begin{bmatrix} 1 & 1 \end{bmatrix} \otimes \left(\begin{bmatrix} I_n \\ -I_n \end{bmatrix}^{\otimes k} \cdot B(\sigma) \right);$$

- $\gamma = \begin{bmatrix} 1 & 0 \end{bmatrix}^\top \otimes e_1^\top$.

We define a set of recognisable tree series $\mathcal{C} := \{\|A\| : A \in \mathcal{A}\}$.

In Lemma 20 we state some properties of the functions in \mathcal{C} . More precisely, we show that the coefficient of a tree $t \in T_\Sigma$ in any series $f \in \mathcal{C}$ fundamentally depends on whether t has 0, 1, or at least 2 nodes whose label is not σ_0 or σ_1 . Here for every $i \in \mathbb{N}_0$ and $t \in T_\Sigma$, we use $\sigma_1^i(t)$ to denote the tree $\underbrace{\sigma_1(\sigma_1(\dots \sigma_1(t) \dots))}_i$.

Lemma 20 *The following properties hold for every $f \in \mathcal{C}$ and $t \in T_\Sigma$:*

- (i) *If $t = \sigma_1^j(\sigma_0)$ where $j \in \{0, 1, \dots, n-1\}$, then $f(\sigma_0) = 1$ and $f(\sigma_1^j(\sigma_0)) = 0$ for $j > 0$.*
- (ii) *If $t = \sigma_1^j(\sigma(\sigma_1^{i_1}(\sigma_0), \dots, \sigma_1^{i_k}(\sigma_0)))$ where $k \in \{0, 1, \dots, m\}$, $\sigma \in \Sigma_k \setminus \{\sigma_0, \sigma_1\}$, and $j, i_1, \dots, i_k \in \{0, 1, \dots, n-1\}$, then $f(t) = B(\sigma)_{(1+i_1, \dots, 1+i_k), (1+n-j) \bmod n}$.*
- (iii) *If $\sum_{\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}} \#_\sigma(t) \geq 2$, then $f(t) = 0$.*

Proof Let $A = (2n, \Sigma, \mu, \gamma) \in \mathcal{A}$ be such that $\|A\| \equiv f$. First, we prove property (i). Using Equation (2) and the mixed-product property of Kronecker product, we get that

$$\mu(\sigma_1^j(\sigma_0)) = \mu(\sigma_0) \cdot \mu(\sigma_1)^j = (\begin{bmatrix} 1 & 0 \end{bmatrix} \otimes e_1) \cdot (I_2 \otimes P^j) = \begin{bmatrix} 1 & 0 \end{bmatrix} \otimes e_1 P^j \quad (17)$$

and therefore

$$f(\sigma_1^j(\sigma_0)) = \mu(\sigma_1^j(\sigma_0)) \cdot \gamma = (\begin{bmatrix} 1 & 0 \end{bmatrix} \otimes e_1 P^j) \cdot (\begin{bmatrix} 1 & 0 \end{bmatrix}^\top \otimes e_1^\top)$$

$$= ([1 \quad 0] \cdot [1 \quad 0]^\top) \otimes (e_1 P^j \cdot e_1^\top) = e_{j+1} \cdot e_1^\top. \quad (18)$$

If $j = 0$ then the expression of (18) is equal to 1, otherwise the expression of (18) is equal to 0. This completes the proof of property (i).

Next, we prove property (ii). By the mixed-product property of Kronecker product and Equations (2), (3), and (17), we have

$$\begin{aligned} & \mu(\sigma_1^j(\sigma(\sigma_1^{i_1}(\sigma_0), \dots, \sigma_1^{i_k}(\sigma_0)))) \\ &= \left(\bigotimes_{l=1}^k \mu(\sigma_1^{i_l}(\sigma_0)) \right) \cdot \mu(\sigma) \cdot \mu(\sigma_1)^j \\ &= \left([1] \otimes \bigotimes_{l=1}^k \mu(\sigma_1^{i_l}(\sigma_0)) \right) \cdot \left([1 \quad 1] \otimes \left(\begin{bmatrix} I_n \\ -I_n \end{bmatrix}^{\otimes k} \cdot B(\sigma) \right) \right) \cdot (I_2 \otimes P)^j \\ &= \left(([1] \cdot [1 \quad 1]) \otimes \left(\bigotimes_{l=1}^k \mu(\sigma_1^{i_l}(\sigma_0)) \cdot \begin{bmatrix} I_n \\ -I_n \end{bmatrix}^{\otimes k} \cdot B(\sigma) \right) \right) \cdot (I_2 \otimes P^j) \\ &= \left([1 \quad 1] \otimes \left(\bigotimes_{l=1}^k \left(([1 \quad 0] \otimes e_1 P^{i_l}) \cdot \begin{bmatrix} I_n \\ -I_n \end{bmatrix} \right) \cdot B(\sigma) \right) \right) \cdot (I_2 \otimes P^j) \\ &= \left([1 \quad 1] \otimes \left(\bigotimes_{l=1}^k e_1 P^{i_l} \cdot B(\sigma) \right) \right) \cdot (I_2 \otimes P^j) \\ &= ([1 \quad 1] \cdot I_2) \otimes \left(\bigotimes_{l=1}^k e_{1+i_l} \cdot B(\sigma) \cdot P^j \right) \\ &= [1 \quad 1] \otimes (B(\sigma)_{(1+i_1, \dots, 1+i_k)} \cdot P^j) \end{aligned} \quad (19)$$

and therefore, using the fact that $P^n = I_n$, we get that

$$\begin{aligned} f(\sigma_1^j(\sigma(\sigma_1^{i_1}(\sigma_0), \dots, \sigma_1^{i_k}(\sigma_0)))) &= \mu(\sigma_1^j(\sigma(\sigma_1^{i_1}(\sigma_0), \dots, \sigma_1^{i_k}(\sigma_0)))) \cdot \gamma \\ &= ([1 \quad 1] \otimes (B(\sigma)_{(1+i_1, \dots, 1+i_k)} \cdot P^j)) \cdot ([1 \quad 0]^\top \otimes e_1^\top) \\ &= ([1 \quad 1] \cdot [1 \quad 0]^\top) \otimes (B(\sigma)_{(1+i_1, \dots, 1+i_k)} \cdot P^j \cdot e_1^\top) \\ &= B(\sigma)_{(1+i_1, \dots, 1+i_k)} \cdot (e_1 P^{n-j})^\top \\ &= B(\sigma)_{(1+i_1, \dots, 1+i_k), (1+n-j) \bmod n}. \end{aligned}$$

Finally, we prove property (iii). If $\sum_{\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}} \# \sigma(t) \geq 2$ then there exists a subtree $\sigma'(t_1, \dots, t_k)$ of t such that $k \geq 1$, $\sigma' \in \Sigma_k \setminus \{\sigma_1\}$, and $\sum_{\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}} \# \sigma(t_i) = 1$ for some $i \in [k]$. It follows from Equation (19) that $\mu(t_i) = [1 \quad 1] \otimes \alpha$ holds for some $\alpha \in \mathbb{F}^{1 \times n}$. By the mixed-product property of Kronecker product and Equation (3), we have

$$\begin{aligned} \mu(\sigma'(t_1, \dots, t_k)) &= \left(\bigotimes_{j=1}^k \mu(t_j) \right) \cdot \left([1 \quad 1] \otimes \left(\begin{bmatrix} I_n \\ -I_n \end{bmatrix}^{\otimes k} \cdot B(\sigma') \right) \right) \\ &= [1 \quad 1] \otimes \left(\bigotimes_{j=1}^k \mu(t_j) \cdot \begin{bmatrix} I_n \\ -I_n \end{bmatrix}^{\otimes k} \cdot B(\sigma') \right) \end{aligned}$$

$$= \begin{bmatrix} 1 & 1 \end{bmatrix} \otimes \left(\bigotimes_{j=1}^k \left(\mu(t_j) \cdot \begin{bmatrix} I_n \\ -I_n \end{bmatrix} \right) \cdot B(\sigma') \right) = 0_{1 \times 2n}$$

where the last equality holds since

$$\mu(t_i) \cdot \begin{bmatrix} I_n \\ -I_n \end{bmatrix} = \begin{bmatrix} \alpha & \alpha \end{bmatrix} \cdot \begin{bmatrix} I_n \\ -I_n \end{bmatrix} = 0_{1 \times n}.$$

Since $\sigma'(t_1, \dots, t_k)$ is a subtree of t , we now have that $\mu(t) = 0_{1 \times 2n}$ and thus $f(t) = 0$. \blacksquare

Remark 21 As $P^n = I_n$, we have $\mu(\sigma_1)^n = I_{2n}$. Thus for every $f \in \mathcal{C}$, $k \in \{0, 1, \dots, m\}$, $\sigma \in \Sigma_k \setminus \{\sigma_0, \sigma_1\}$, and $j, i_1, \dots, i_k \in \mathbb{N}_0$, it holds that $f(\sigma_1^j(\sigma_0)) = f(\sigma_1^{j \bmod n}(\sigma_0))$ and

$$f(\sigma_1^j(\sigma(\sigma_1^{i_1}(\sigma_0), \dots, \sigma_1^{i_k}(\sigma_0)))) = f(\sigma_1^{j \bmod n}(\sigma(\sigma_1^{i_1 \bmod n}(\sigma_0), \dots, \sigma_1^{i_k \bmod n}(\sigma_0)))).$$

Run **L** on a target $f \in \mathcal{C}$. Lemma 20 (i), (iii) and Remark 21 imply that when **L** makes a membership query on $t \in T_\Sigma$ such that $\sum_{\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}} \#_\sigma(t) \geq 2$, the Teacher returns 0, while when **L** makes a membership query on $t = \sigma_1^j(\sigma_0)$, the Teacher returns 1 if $j \bmod n = 0$ and returns 0 otherwise. In these cases, **L** does not gain any new information about f since every function in \mathcal{C} is consistent with the values returned by the Teacher.

When **L** makes a membership query on a tree $t = \sigma_1^j(\sigma(\sigma_1^{i_1}(\sigma_0), \dots, \sigma_1^{i_k}(\sigma_0)))$ such that $k \in \{0, 1, \dots, m\}$ and $\sigma \in \Sigma_k \setminus \{\sigma_0, \sigma_1\}$, the Teacher returns an arbitrary number in \mathbb{F} if the value $f(t)$ is not already known from an earlier query. Lemma 20 (ii) and Remark 21 imply that **L** thereby learns the entry $B(\sigma)_{(1+(i_1 \bmod n), \dots, 1+(i_k \bmod n), (1+n-j) \bmod n)}$.

When **L** makes an equivalence query on a hypothesis $h \in \mathcal{C}$, the Teacher finds some entry $B(\sigma)_{(i_1, \dots, i_k), j}$ that **L** does not know from previous queries and returns the tree $\sigma_1^{1+n-j}(\sigma(\sigma_1^{i_1-1}(\sigma_0), \dots, \sigma_1^{i_k-1}(\sigma_0)))$ as the counterexample.

With each query, the Learner **L** learns at most one entry of $B(\sigma)$ where $\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}$. The number of queries made by **L** on target f is, therefore, at least the total number of entries of $B(\sigma)$ for all $\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}$. The latter number is equal to

$$\begin{aligned} \sum_{\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}} n^{rk(\sigma)+1} &\geq \frac{1}{2^{m+1}} \cdot \sum_{\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}} r^{rk(\sigma)+1} \\ &= \frac{1}{2^{m+1}} \cdot \left(\sum_{\sigma \in \Sigma} r^{rk(\sigma)+1} - r^2 - r \right). \end{aligned}$$

\blacksquare

The lower bounds of Theorems 18 and 19 are both linear in the target automaton size. Note that when the alphabet rank is fixed, the lower bound for learning over a fixed field (Theorem 19) is the same up to a constant factor as for learning over an arbitrary field (Theorem 18).

Assuming a Teacher that represents counterexamples as succinctly as possible, e.g, using the algorithm of Seidl (1990), the upper bound of algorithm **LMTA** from Theorem 17 is quadratic in the target automaton size, i.e., quadratically greater than the lower bound of Theorem 18.

6. Future Work

Beimel et al. (2000) apply their exact learning algorithm for multiplicity word automata to show exact learnability of certain classes of polynomials over both finite and infinite fields. They also prove the learnability of disjoint DNF formulae (i.e., DNF formulae in which each assignment satisfies at most one term) and, more generally, disjoint unions of geometric boxes over finite domains.

The learning framework considered in this paper involves tree automata, which are more expressive than word automata. Moreover, our result on the complexity of equivalence of multiplicity tree automata shows that, through equivalence queries, the Learner essentially has an oracle for polynomial identity testing. Thus a natural direction for future work is to seek to apply our algorithm to derive new results on exact learning of other concept classes, such as propositional formulae and polynomials (both in the commutative and noncommutative cases). In this direction, we would like to examine the relationship of our work with that of Klivans and Shpilka (2006) on exact learning of algebraic branching programs and arithmetic circuits and formulae. The latter paper relies on rank bounds for Hankel matrices of polynomials in noncommuting variables, obtained by considering a generalised notion of partial derivative. Here we would like to determine whether the extra expressiveness of Hankel matrices over tree series can be used to show learnability of more expressive classes of formulae and circuits.

Sakakibara (1990) showed that context-free grammars (CFGs) can be learned efficiently in the exact learning model using membership queries and counterexamples based on parse trees. Given the important role of weighted and probabilistic CFGs across a range of applications including linguistics, another natural next step would be to apply our algorithm to learn weighted CFGs similarly using queries and counterexamples involving parse trees.

Acknowledgments

The authors would like to thank Michael Benedikt for stimulating discussions and helpful advice. The first author gratefully acknowledges the support of the EPSRC.

References

- A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.
- S. Anantharaman, P. Narendran, and M. Rusinowitch. Closure properties and decision problems of DAG automata. *Information Processing Letters*, 94(5):231–240, 2005.
- D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
- D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

- S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
- R. Bailly, F. Denis, and L. Ralaivola. Grammatical inference as a principal component analysis problem. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 33–40, 2009.
- B. Balle and M. Mohri. Spectral learning of general weighted automata via constrained matrix completion. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2168–2176, 2012.
- A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *J. ACM*, 47(3):506–530, 2000.
- J. Berstel and C. Reutenauer. Recognizable formal power series on trees. *Theoretical Computer Science*, 18(2):115–148, 1982.
- L. Bisht, N. H. Bshouty, and H. Mazzawi. On optimal learning algorithms for multiplicity automata. In *Proceedings of the 19th Annual Conference on Learning Theory (COLT)*, volume 4005 of *LNCS*, pages 184–198. Springer, 2006.
- S. Bozapalidis and A. Alexandrakis. Représentations matricielles des séries d’arbre reconnaissables. *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications*, 23(4):449–459, 1989.
- S. Bozapalidis and O. Louscou-Bozapalidou. The rank of a formal tree power series. *Theoretical Computer Science*, 27(1):211–215, 1983.
- W. Charatonik. Automata on DAG representations of finite trees. Research Report MPI-I-1999-2-001, Max-Planck-Institut für Informatik, Saarbrücken, 1999.
- H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag, 1993.
- R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978.
- F. Denis, M. Gybels, and A. Habrard. Dimension-free concentration bounds on Hankel matrices for spectral learning. In *Proceedings of the 31th International Conference on Machine Learning (ICML)*, pages 449–457, 2014.
- F. Drewes and J. Högberg. Learning a regular tree language from a teacher. In *Developments in Language Theory*, volume 2710 of *LNCS*, pages 279–291, 2003.
- B. Fila and S. Anantharaman. Automata for analyzing and querying compressed documents. Research Report RR-2006-03, LIFO, 2006.
- M. Gybels, F. Denis, and A. Habrard. Some improvements of the spectral learning approach for probabilistic grammatical inference. In *Proceedings of the 12th International Conference on Grammatical Inference (ICGI)*, pages 64–78, 2014.

- A. Habrard and J. Oncina. Learning multiplicity tree automata. In *Proceedings of the 8th International Colloquium on Grammatical Inference: Algorithms and Applications (ICGI)*, volume 4201 of *LNCS*, pages 268–280. Springer, 2006.
- S. Kiefer, A. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. On the complexity of equivalence and minimisation for \mathbb{Q} -weighted automata. *Logical Methods in Computer Science*, 9(1), 2013.
- A. R. Klivans and A. Shpilka. Learning restricted models of arithmetic circuits. *Theory of Computing*, 2(1):185–206, 2006.
- I. Marusic and J. Worrell. Complexity of equivalence and learning for multiplicity tree automata. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS), Part I*, volume 8634 of *LNCS*, pages 414–425. Springer, 2014.
- Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76(2-3):223–242, 1990.
- M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.
- J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.
- H. Seidl. Deciding equivalence of finite tree automata. *SIAM J. Comput.*, 19(3):424–437, 1990.
- L. G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI), Volume 1*, pages 560–566. Morgan Kaufmann, 1985.
- R. E. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (EUROSAM '79)*, volume 72 of *LNCS*, pages 216–226. Springer, 1979.